Section CONTENTS Page

APPENDICES

1 Introduction to Extension Basic

Extension Basic was produced by, and is Copyright (C) of Level 9 Computing: 229 Hughenden Road, High Wycombe, Bucks.

Extension Basic is a program which adds extra commands and statements to Nascom ROM BASIC (Microsoft 4.7), greatly enhancing this. The new commands and statements are used as if they were an integral part of ROM BASIC.

Extension Basic (EB) is fully compatible with ROM BASIC: all of your existing programs can be run or modified using it (unless they POKE into memory used by EB, of course). Similarly, programs can be produced using EB and then run using ROM BASIC alone (although any EB keywords will, naturally, give syntax errors). It is very simple to switch between the two BASICs.

Thus EB is both:
- a 12K BASIC interpreter for the Nascom;
- a very up-market toolkit for producing programs to run under ROM BASIC.

This manual describes how to use EB, what it does, and includes some technical details to help you get the best from it.

2 How to Use Extension Basic

Extension Basic is supplied on cassette, packaged within a relocator program. A demonstration EB program follows it on the cassette, which is described in Appendix 1. In addition, if you have bought EB in ROM, you will have 2*2716 or 4*2708 ROMs containing the EB interpreter.

When you first use EB from cassette, it is necessary to load the relocator program and execute it to place the EB interpreter at a suitable memory address. It is advisable to position it at the top of your Nascom's memory. This process is described in Appendix 3. Then you could save the generated EB on cassette, so that on subsequent occasions you could load it from there.

When you first use EB in ROM, plug in the ROMs as outlined in Appendix 3.

Starting and restarting Extension Basic are described fully in Appendix 4. To initialise it, you enter:

J	(to initialise ROM BASIC, using a memory size that will prevent it overwriting EB);
MONITOR	(to return to Nas-Sys):
E address	(where address is the lowest address of EB: its start address).

To start or restart it, once initialised, enter:

Z	(to restart BASIC);
SET	(to switch to using Extension Basic).

Once started, Extension Basic is just like Nascom ROM BASIC, but with many extra commands and statements.

Step-by-step instructions for relocating and starting Extension Basic, and for using the demonstration program are included in Appendix 2. Follow them if this will help you.

3 New Statements

3.1 Summary

Extension Basic provides the following commands/statements in addition to all those of ROM BASIC. Each can be used either as a command or a statement, although some are better suited to one role than the other. This section summarises all the additional facilities; they are described later.

Page	Keyword	Parameters	Description
4	AUTO	lnum,increment	Automatically number newly entered lines;
12	BREAK	+	Enable/disable break action of ESC key;
12	CALL	address,list	Call 'USR' with multiple arguments;
5	CHECK		Check for undefined refences (UL errors);
9	COPY	from,to,length	Copy block of memory;
6	DEC	hex-number	Print decimal equivalent of hex number;
12	DELAY	value	Do nothing for value msec;
5	DELETE	lnum,lnum	Delete line(s);
4	EDIT	lnum	Edit long lines;
6	FIND	string	Find line(s) containing string;
8	GET	variable	Return ascii code of next key pressed (or ø if no key has been pressed);
6	HEX	decimal-number	Print hex equivalent of decimal number;
11	IF..THEN..ELSE		Either do one thing or another;
8	INKEY	variable	Wait for next key to pressed and return its ascii code;
8	INLIN	string-variable	Return entire screen line containing cursor when ENTER is pressed;
9	LINE	x1,y1,x2,y2,type	Set/reset/reverse graphics line;
12	LIST	lnum	LIST, showing graphics characters properly
9	PLOT	x,y,type	Set/reset/reverse pixel;
10	PRINT	@x,etc	Print, starting at address on screen;
10	PUT	list	Print strings and characters;
11	REPEAT..UNTIL		Repeatedly loop until condition true;
5	RENUMBER	lnum,increment	Renumber program lines;
5	REDUCE	code	Reduce size of program: removes spaces and comments;
12	SET	(-)	Initialise EB, Check EB is running;
7	SPEED	value,value	Set keyboard repeat speed (msec);
8	TEST	code,result	Check if specified key is held down;
7	TRACE	code,string	Trace line numbers and specified variable values during program execution, optionally slowed down;
10	VDU	x,y,string	Print string at address (even on top line), and string expression can be printed;
11	WHILE..WEND		If condition true, loop while it is true;
10	WRAP	+	Enable/disable word wrap-around;
7	XLIST	lnum	List line(s) plus references to them;
7	XREF	lnum	List numbers of lines referencing specified line.

3.2 Editing Commands

These commands are intended to make the work of producing programs easier. They are best used in direct mode (ie following a prompt of OK) rather then as statements within a program.

3.2.1 AUTO lnum,increment

Example:
AUTO 10,10

This command automatically supplies a number for each new program line entered. The line number is printed, followed by a cursor right, and the cursor stops. You then type the text of the program line and press 'ENTER', whereapon the line is entered as usual and a number for the next line printed. And so on..

To stop AUTO, press 'ENTER' when the cursor is on a line which does not begin with a line number, eg when it is on a blank line.

The first line number supplied is lnum. Each subsequent number is calculated by adding increment to the number of the entered line, and it is allright to change the line number supplied automatically: the line is entered with the amended number and the next line number is increment plus this.

3.2.2 EDIT lnum

Example:
EDIT 10

This command allows very long lines (up to the length that will fit on the entire screen - about 700 characters) to be entered and modified.

The screen is cleared and the selected line listed on it. The line can then be modified as under Nas-Sys, except that the entire screen is treated as a single line. Pressing 'ENTER' enters the line, clears the screen and leaves EDIT - listing the line again so that you can check that it is correct.

To abandon any changes made and leave EDIT, either:
- use SHIFT ← to clear the screen, character by character, and press 'ENTER';
or press RESET and then restart EB by typing Z and SET.

If word wrap is set when you use EDIT, odd but predictable side effects can occur: especially if you are sparing in your use of spaces in statements.

The following characters behave differently in EDIT than in the normal Nas-Sys edit mode:

ESC	appears as ⊖
CS	appears as ↡
SHIFT ←	shifts all characters left, up to end-of-screen
SHIFT →	shifts all characters right, up to end-of-screen
CTRL J	appears as ☰
CTRL X	appears as ⧗

3.2.3 DELETE lnum,lnum

Example:
```
DELETE 190,230
```

All lines between and including the specified lines are deleted from the program. Nothing is done if the second number is smaller than the first.

3.2.4 RENUMBER lnum,increment

Example:
```
RENUMBER 10,10
```

The program is renumbered so that its first line has number lnum and subsequent lines have numbers increasing by increment. All GOTO, GOSUB, ON and RESTORE statements are altered to reference the new line numbers.

The program is checked before any alterations are made, and if any statement references a line which does not exist then RENUMBER stops with an error and the program is not changed in any way. Similar checks are made to ensure that line numbers will not be to big after renumbering and to guard against running out of memory if the program gets bigger.

Renumber takes about 40 seconds for the Level 9 FANTASY program which is 16K in size and is 470 lines in length (with several statements per line).

3.2.5 CHECK

Example:
```
CHECK
```

CHECK carries out the same checking for undefined references as does RENUMBER, but does not renumber the program.

3.2.6 REDUCE

Example:
```
REDUCE 3
```

REDUCE makes a program smaller. It has 3 modes of action:

- REDUCE 1 removes all spaces from the program, except for those in REM or DATA statements, or within quotes;
- REDUCE 2 removes all REM statements, except for those at the start of lines, in which case the REM is left but all following text is removed;
- REDUCE 3 acts as REDUCE 1 and REDUCE 2 together.

3.3 Debugging Commands

These commands are intended to reduce the amount of work required to solve bugs in a program. They are best used in direct mode (ie following a prompt of OK), rather than as statements in a program.

3.3.1 DEC hexadecimal-number

Example:
```
DEC 10D6
```

This command prints the decimal equivalent of a hexadecimal number.

3.3.2 FIND string-expression

Examples:
```
FIND "THEN GOSUB 1000"

F$="A=": REM Set up search string for later use.
FIND F$+"0": REM Find A=0
```

FIND is followed by a string, or a string expression. When 'ENTER' is pressed. the string is stored and the program is scanned to find lines containing it. The text is found wherever it is in the lines (in strings. in REM statements. as keywords etc).

Each line found is displayed on the bottom line of the screen, scrolling higher lines. FIND then pauses and flashes the cursor, waiting for ESC to quit or any other key to continue. FIND stops when the entire program has been scanned.

In technical terms, FIND evaluates its argument as a string-expression and then compresses keywords in the same way as does BASIC. Then it scans the program comparing the compressed string against the compressed text maintained by BASIC. Normally you can forget about this, but it does mean that, for example, FIND "RE" will not find REM statements.

3.3.3 HEX decimal-number

Example:
```
HEX 32
```

This command prints the hexadecimal equivalent of a decimal number.

3.3.4 SPEED delay,interval

Example:
```
SPEED 200,100
```

This sets the speed of the built-in repeat keyboard, which repeats any key that you hold down. Delay is the approx number of msec before repetition starts and interval is the approx number of msec between repetitions. To effectively turn repeat keyboard off, enter SPEED 0,0.

The SPEED command takes over from the Nas-Sys 3 repeat keyboard feature, if you have this, and there is no conflict.

3.3.5 TRACE code,string-expression

Example:
```
TRACE 1,"X;Y;X(1):X(2):X(3):A$"
```

TRACE with a code greater than 0 results in the number of each line executed being printed, together with the contents of any variables named in the string-expression following.

TRACE 0	turns off tracing;
TRACE 1,string-expression	traces each line, pausing until a key is pressed;
TRACE n,string-expression	traces each line, with a delay of approx n msec.

3.3.6 XLIST lnum

Example:
```
XLIST 110
```

This behaves like the normal LIST command, but for each line listed it prints out the numbers of all lines which contain references to it. E.g:

```
110 INKEY C: IF C<48 OR C>57 THEN PRINT "That is not a number ";: GOTO 110
<20><110><210>
```

3.3.7 XREF lnum

Example:
```
XREF 110
```

This prints out the numbers of all lines which contain references to the specified line

3.4 Input Statements

These statements allow input from the keyboard in ways suited to interactive, user-friendly programs.

3.4.1 GET variable

Example:
```
GET C
```

Returns the ASCII code of any key pressed, or 0 if no key has been pressed. GET does not wait for a key to be pressed. If several keys are pressed, then successive calls to GET return the value of each key in turn. Get uses the Nas-Sys routine 'IN'.

3.4.2 INKEY variable

Example:
```
INKEY C
```

Returns the ASCII code of the next key to be pressed, waiting for a key to be pressed and flashing the cursor. Inkey uses the Nas-Sys routine 'BLINK'.

3.4.3 INLIN string-variable

Example:
```
INLIN L$
```

Returns the entire line containing the cursor when 'ENTER' is pressed. The screen can be modified and the cursor can be moved while INLIN is waiting for the 'ENTER'. Trailing spaces are removed from the input line.

3.4.4 TEST code,result

Example:
```
TEST 52,R: IF R THEN GOSUB 1000
```

Tests to see whether the key specified by code is actually up or down, returning a result of 0 for up, or 1 for down. The value of code is NOT the ASCII value for the key, but a hardware related number (see Appendix 9 for details).

3.5 Screen Handling Statements

These statements will be useful for programs which make use of the screen for graphics, interactive games etc.

3.5.1 COPY from,to,length

Example:
```
1 COPY 2058,2122,896
```

This copies a block of memory, whose length in bytes is given by the third argument from the address given by the first argument to the address given by the second argument. This is useful for moving the screen etc.

No matter how the from and to areas overlap, the copy operation will occur without side effects as copying is done from right or left to avoid ever copying from a byte after it has been copied to.

Warning: So that COPY is as versatile as possible, the from and to addresses are not checked. Be careful you do not copy to the middle of your program source unless you want to for some reason.

3.5.2 LINE x1,y1,x2,y2,code

Example:
```
1 LINE 0,0,95,44,1
```

This draws a line of pixels on the screen from x1,y1 to x2,y2. Coordinates are as for SET and RESET, eg: 0,0 is the top left; 95,44 is the bottom right; and 0,22 is the middle of the leftmost side of the screen.

If code=0 then all points on the line are RESET, for a code of 1 they are SET, and for a code of 2 they are reversed.

Note that the algorithm used by LINE is quite clever but is optimised for speed, and the following lines are not necessarily exactly the same for arbitrary coordinates (they match for diagonal or orthogonal lines, however):
```
1 LINE A,B,U,V,1
2 LINE U,V,A,B,1
```

3.5.3 PLOT x,y,code

Example:
```
1 PLOT 0,0,2
```

If code=0 the point x,y is RESET, if code=1 it is SET, and if code=2 it is reversed. Coordinates are as for SET, RESET and LINE.

10

3.5.4 PRINT @screen-address:etc

Example:
```
1 PRINT @10;"*": REM Print asterisk at top left of the screen.
```

If the first non-space character after a PRINT is an @, then the cursor is set to the screen-address following the @, before the remaining arguments of the PRINT are output as normal. This is to help with conversion of BASIC programs written for other computers such as the TRS80 for the Nascom.

3.5.5 PUT list-of-expressions

Example:
```
1 PUT 12,"A message on a blank screen"
```

PUT prints a list of arguments, which follow it separated by commas. An argument which evaluates to a string is printed. An argument which evaluates to a number is converted to the equivalent ASCII character and then printed.

3.5.6 WRAP + or WRAP -

Example:
```
1 WRAP + enables word-wraparound and WRAP - disables it again.
```

Word-wraparound prevents any word being printed half on one line and half on the next by wrapping it onto the following line. This greatly simplifies programs which output text and is especially good for word processing programs such as editors, ADVENTURE, ELIZA or FANTASY.

Warning: word-wraparound continues in effect in direct mode and can have odd but predictable side effects when lines are LISTed, especially if you are sparing in your uses of spaces in statements.

3.5.7 VDU X,Y,string

Example:
```
1 VDU 10,16,"Max Score = "+STR$(M)
```

VDU prints a string expression starting at the character coordinates supplied, even on the top line. Coordinates are as for SCREEN.

3.6 Structured Statements

You need never use another GOTO.

3.6.1 IF condition THEN statement(s) : ELSE statement(s)

Example:
```
1 IF X=0 THEN Z=9999: ELSE Z=Y/X
```

IF statements which are followed by an ELSE on the same line work as follows:
- If the condition is true, then the statement(s) between the THEN and the ELSE are executed, or if a line number is there it is gone to.
- If the condition is false, then the statement(s) after the ELSE are executed.

The colon before the ELSE must be present.

IF statements which are not followed by ELSE work as before.

3.6.2 REPEAT ... UNTIL condition

Example:
```
1 REPEAT
2 INKEY C
3 UNTIL C=32
```

The statements between a REPEAT and the corresponding UNTIL are repeated (at least once) until the condition is true.

REPEAT starts a repeat-until loop. UNTIL ends it and must be followed by an expression. Ths REPEAT and the UNTIL can be on the same line, on different lines, nested etc. See Appendix 6 for details of nesting.

3.6.3 WHILE condition ... WEND

Example:
```
1 INPUT "How many bases";B
2 WHILE B 0
3 GOSUB 1000: REM Play game
4 WEND
```

WHILE starts a while loop. If the condition following it is true, the statements between it and the corresponding WEND statement are repeatedly executed until the condition becomes false.

WEND ends a while loop.

WHILE and WEND must both be at the start of lines, for efficiency reasons.

While loops can be nested with other loops (see Appendix 6 for details).

12

3.7 General Statements

The following general-purpose statements are provided:

3.7.1 BREAK + or BREAK -

Example:
```
1 BREAK -
```

BREAK - disables the break action of the ESC key, BREAK + enables it.

3.7.2 CALL address,optional-expressions

Example:
```
1 CALL 3200
```

Call a machine code subroutine at the address given by the first argument, passing any number of arguments from none upwards. See Appendix 10 for details.

3.7.3 DELAY delay

Example:
```
1 FOR X=0 TO 64*15: PRINT @X;"*";: DELAY 100: NEXT X
```

Do nothing for a specified number of msec (approx). Delay must be positive and greater than zero.

3.7.4 LIST lnum

Example:
```
LIST 10
```

LIST line(s) as normal, but with graphic characters appearing as themselves, not as keywords. Keywords are shown as keywords, of course.

3.7.5 SET or SET -

Example:
```
SET
```

SET switches from ROM BASIC mode to EB mode. SET - does the same without displaying the "Copyright (C) 1982 Level 9 Computing" message.

SET followed by a pair of coordinates works as before.

13

Appendix 1: Contents of Extension Basic Cassette

The Extension Basic release cassette contains 2 programs:

1) The Extension Basic interpreter packaged with its Relocator program to form an 8K machine code program which will load at 1000 and upwards;
2) The EB Demonstration Program, an 8K program written in Extension Basic to illustrate some of the features of EB and allow you to test that you have installed it properly. It includes 'one line' versions of the games Sweeper and Demon Driver but these are by no means representative of the standard of other games sold by L9.

Each side of the cassette contains one copy of each program, in the order EB, Demo. Side A is recorded at 1200 baud and side B at 300 baud.

The programs follow each other closely and are recorded in standard Nas-Sys format. If you have Q-DOS, you can load them straight onto disk, using the headers, otherwise you should just use the Nas-Sys R command.

Note: do not rush ahead and load EB, expecting it to work instantly. You need to relocate it at a convenient place in memory first, as described in Appendix 3.

If you have purchased EB in ROM, you will also have 2*2716 or 4*2708 ROMs containing the EB interpreter. You should plug these in at the address in memory stated with your order and, in this case, EB should work instantly. See Appendix 3 for more details.

14

Appendix 2: Running the Demonstration

This demonstration takes you in a step-by-step manner through all steps necessary to install Extension Basic from cassette, and use it to run the demonstration EB program supplied.

First turn on your Nascom, cassette deck etc. You will see a message on the tv screen: Nas-Sys 1 or Nas-Sys 3.

Enter R and play the EB cassette in your cassette deck to read in the EB interpreter. (Choose the side of the cassette with the same recording rate as you normally use: side A = 1200 baud; side B = 300 baud).

As the program loads from tape, Nas-Sys displays a descending sequence of numbers. If any of these is followed by a question mark, then an error has occurred in loading: stop the tape, rewind it a little way (past the point at which the error occurred) and continue playing. Eventually the numbers will reach 0 and the cursor will start blinking on the next line of the screen. This means that EB is loaded, so stop the tape.

Now you need to relocate (copy) the Extension Basic interpreter to a place in memory where it will not get in the way. Enter E 1000 and the EB relocator program will ask you for a New Start Address? Enter 4000 to relocate the Extension Basic interpreter to a suitable place (or see Appendix 3 for full details).

Now you could save EB on a cassette of your own, to avoid the need to relocate it when you next want to use it. See Appendix 3 for details.

Enter J to start Nascom ROM BASIC, and this will ask you for a Memory Size? Enter 16385 to specify a size that will prevent BASIC overwriting EB.

Now enter MONITOR to return to Nas-Sys and enter E 4000 to initialise Extension Basic. Finally enter Z to start it. Extension Basic is now available for use.

To load the EB demonstration program, enter CLOAD and play the tape. Deal with any errors (shown by question marks) as above, and stop the tape when the demonstration program has loaded.

Finally, type LIST to examine the demo program, or enter RUN to start it. It will display instructions telling you how to use it.

When you have finished, turn everything off.

15

Appendix 3: Installing Extension Basic

Extension Basic is supplied on cassette or in ROM. This page deals with installing the cassette version, for ROM see the next page.

Extension Basic is supplied as the first program on each side of the cassette. It is packaged within a relocator program.

You must load this and execute the relocator so that the EB interpreter is copied to a convenient location in memory. Then it will be ready for use.

To load the relocator/interpreter into addresses 1000 to 2FFF, enter a single Nas-Sys R command and play the tape as usual.

Then start the relocator by entering E 1000

It will ask you for the start address for the Extension Basic interpreter, in hexadecimal. We suggest that you place it at the very top of memory, so for a 16K Nascom and allowing 4K for Extension Basic, you would enter 4000.

Relocation takes much less than a second.

At this point you might like to save a copy of the relocated program, so that in future you need only load it, without going through the relocation process. The EB interpreter is exactly 4K (1000 hex) in length.

Note that we would consider it a breach of copyright if you made too many copies: one or two for backup purposes would be all right.

Suggested start addresses for Extension Basic on computers which have various memory sizes follow. The sizes are those needed for the BASIC J command, to avoid it overwriting the EB interpreter when ROM BASIC is cold started (see Appendix 4). The bytes free are replies from the J command. Note: if you have ROM EB, you're lucky: there is no danger of overwriting so you don't need to give a number to the J command.

Computer Memory	EB Start Address	J Memory Size	Bytes Free
16K	4000	16385	11971
24K	6000	24577	20163
32K	8000	32769	28355
40K	A000	40961	36547
48K	C000	49153	44738

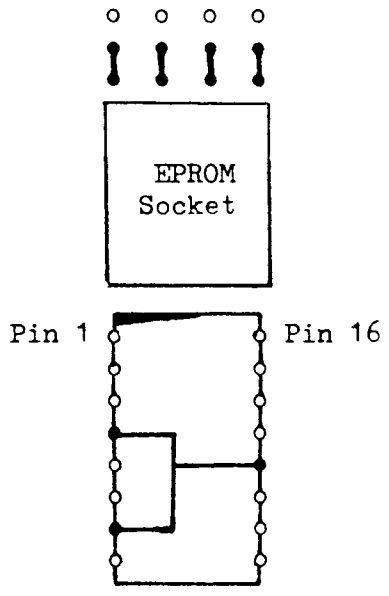
Because of the existence of several Nascom variants, and the fact that you may have already added ROM in the most convenient locations for this, it is difficult to give definite advice on installing the ROM version of EB, except to say: "Read your Nascom Hardware Manual" and "Be careful".

This appendix covers installing EB in 4*2708 EPROMS at a start address of B000, which is one of the simplest cases.

EB is supplied in four 2708 EPROMS labelled EB-1,for the lowest 1K, up to EB-4 for the highest. A release note with the EPROMS should give a tabulated listing of the first few bytes of each chip, to help you check that you have wired the address decoding properly.

Stage 1

For the 4 sockets labelled IC-35 to IC-38, wire the corresponding link blocks as shown.....→



Stage 2

Wire the header plug LKS-1 by adding the following links in addition to those already there.....↘

Stage 3

Make sure that switches LSK1/7 (violet) and LSK1/8 (grey) are in the down position.

Stage 4

Insert the EPROMS, checking their orientation and making sure that their legs do not get damaged.

Put EB-1 in socket A1, IC-35;
and EB-2 in socket A2, IC-36;
and EB-3 in socket A3, IC-37;
and EB-4 in socket A4, IC-38.

Stage 5

Switch on and check that your Nascom works as before. Then use the Nas-Sys T command to display the contents of the first few bytes of each EPROM and compare this with the listing provided. For the final test, execute EB and use it to run the demonstration program.

Appendix 4: Starting Extension Basic

Extension Basic should treated as a 12K BASIC interpreter which requires different initialisation from Nascom ROM BASIC.

Cold Start

To cold start Extension Basic, assuming that it is loaded, follow the sequence below to initialise everything and clear out any program.

1) Cold start ROM BASIC with a suitable memory size (see Appendix 3):

```
J
Memory Size? 16385

NASCOM ROM Basic Ver 4.7
Copyright (C) 1978 by Microsoft
11971 Bytes Free
```

2) Return to Nas-Sys:

```
MONITOR
```

3) Initialise EB itself, by executing it at its first address (see Appendix 3):

```
E 4000
```

4) Finally, warm start BASIC (as RESET has not been pressed, this leaves you in Extension Basic mode):

```
Z
```

Warm Start

If you press RESET, then entering Z leaves you using ROM BASIC alone. To enter Extension Basic mode, use the SET command, eg:

```
Z
Ok
SET

Extension Basic 1.2
Copyright (C) 1982 Level 9 Computing

Ok
```

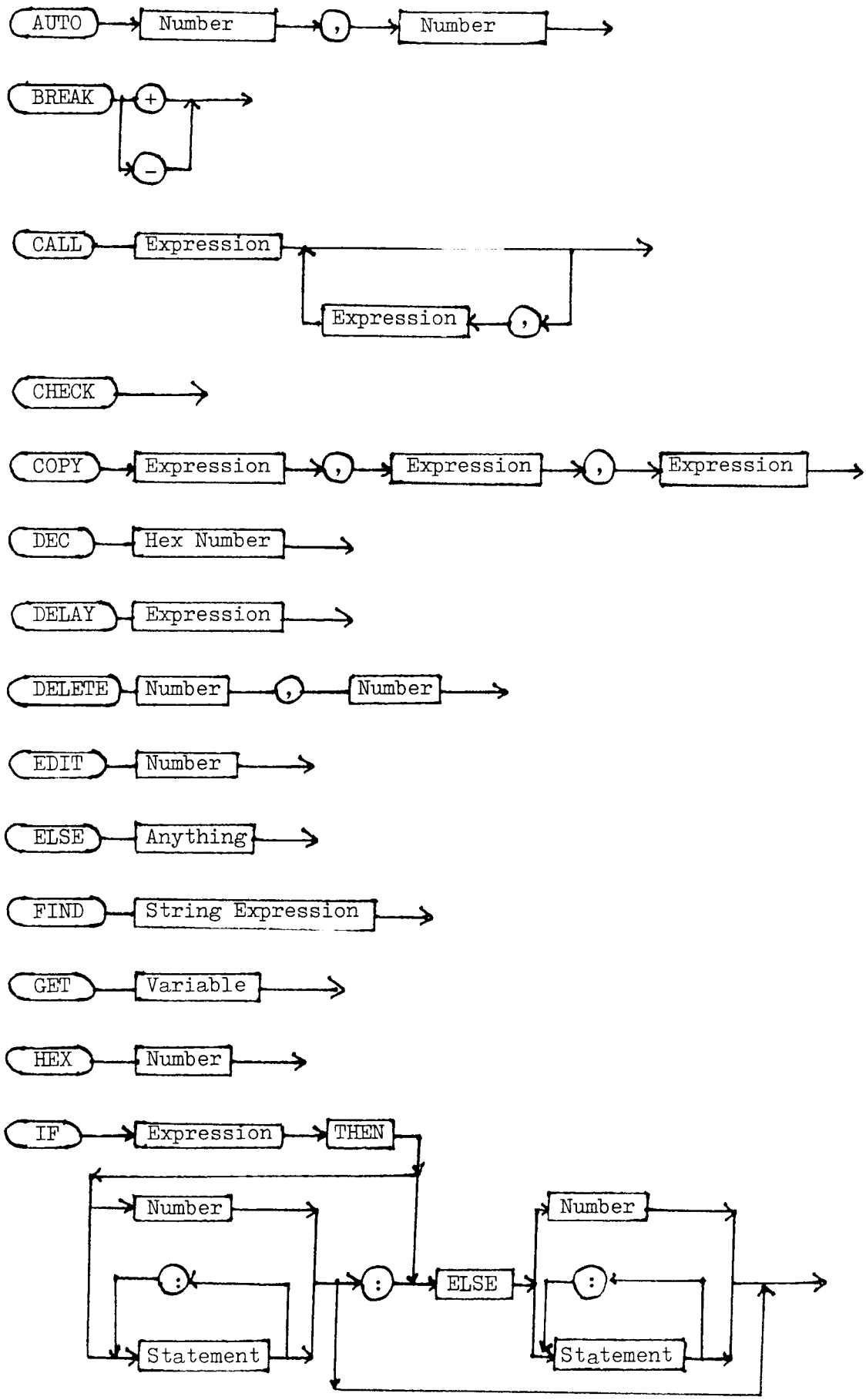
An alternative way of warm starting Extension Basic from Nas-Sys is to enter:

```
E 1000
```

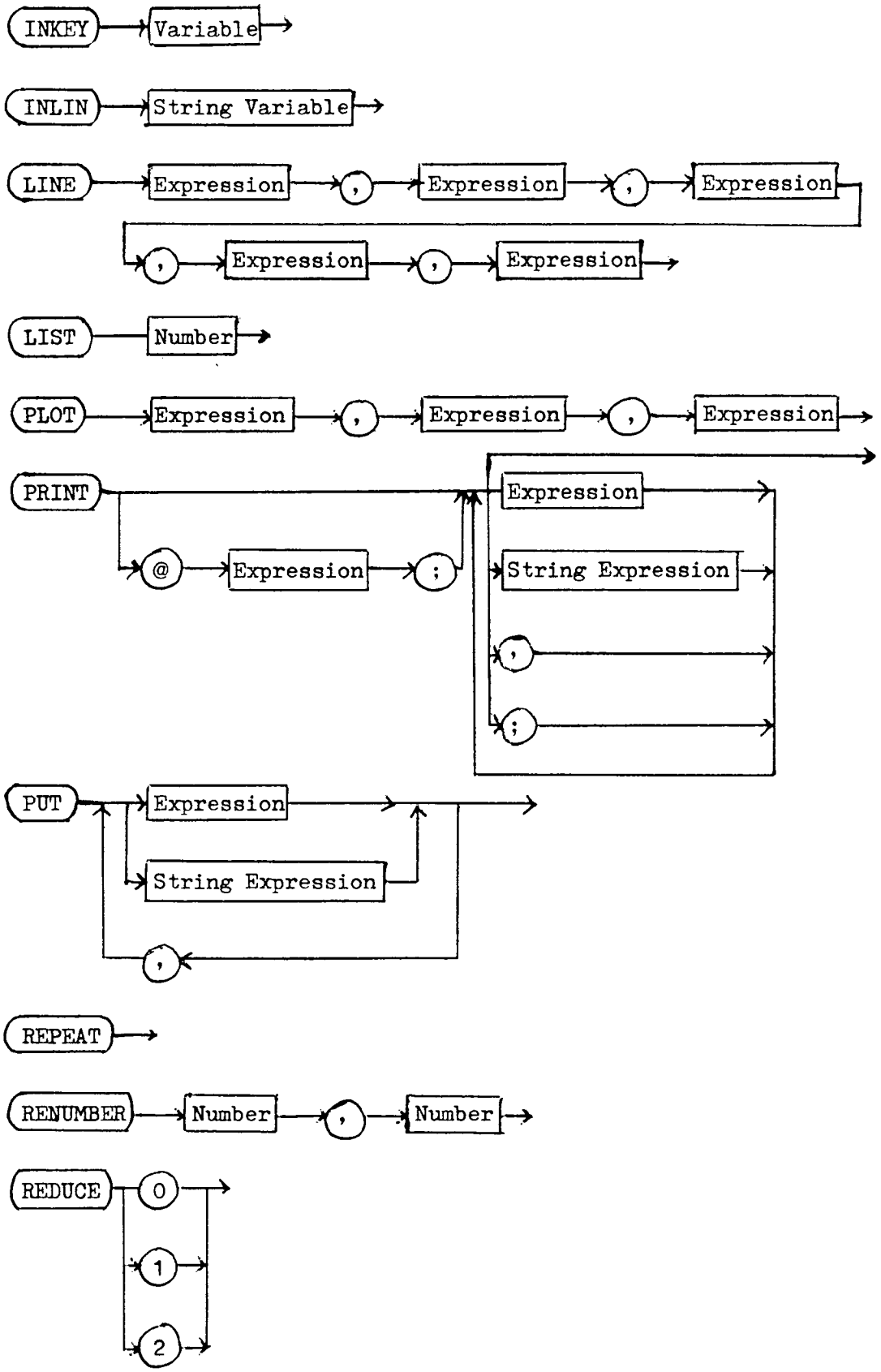
Automatic Cold Start

If you have bought the ROM version of EB, you can arrange for the power-on-reset jump of the Nascom 2 to automatically initialise it before returning to Nas-Sys for you to initialise ROM BASIC (see your Nascom hardware manual: you need to set the jump address to the first address of EB). If you additionally ensure that the first instruction of your BASIC programs is SET - you will in effect be able to use EB as if you were just using ROM BASIC: EB will be initialised and then started automatically. Note: the code for this initialisation takes the place of the checksum code in the RAM version, and is the same size exactly.

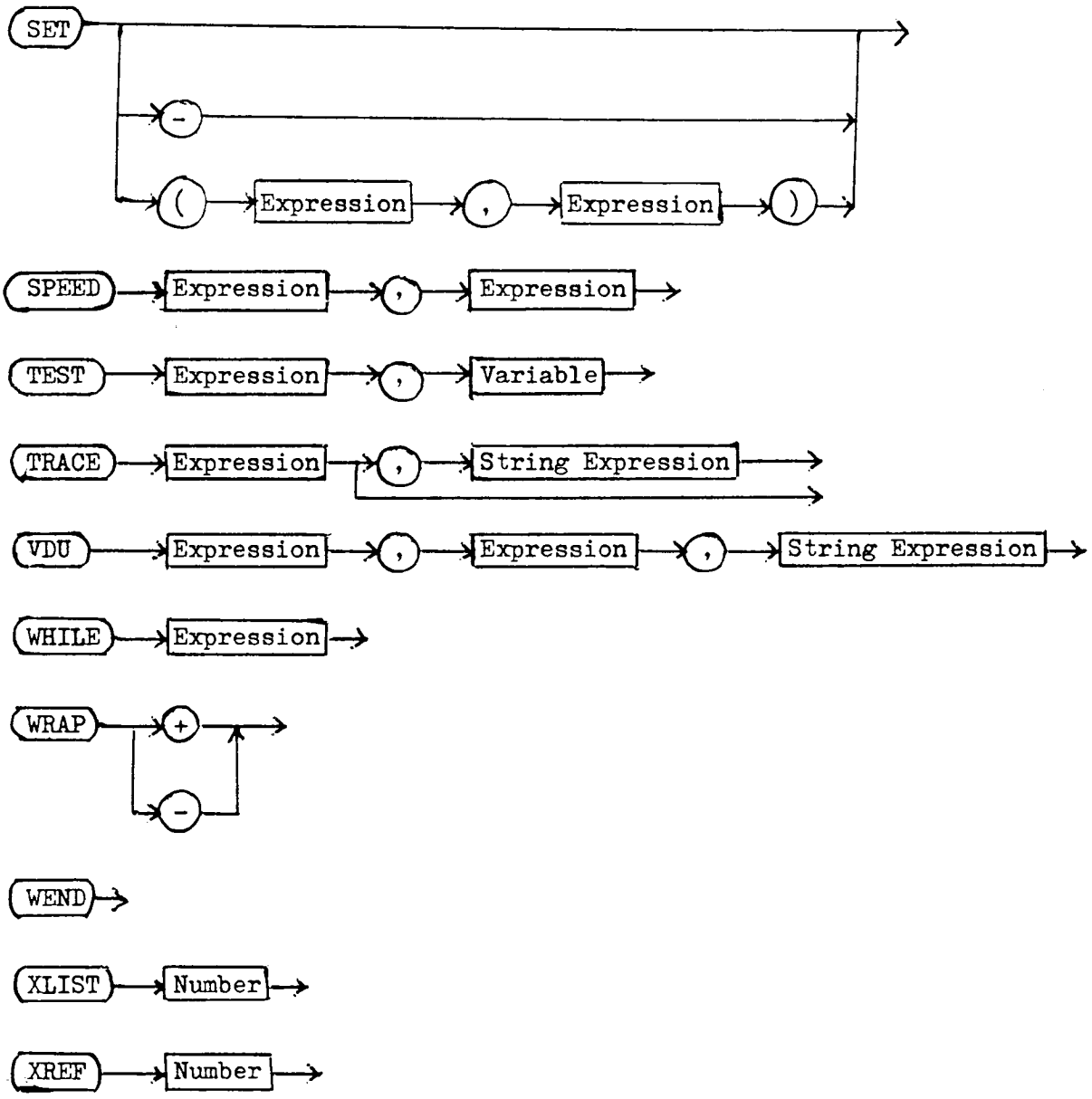
Appendix 5: Syntax of Statements



19


INKEY
Variable
INLIN
String Variable
LINE
Expression
,
Expression
,
Expression
,
Expression
,
Expression
LIST
Number
PLOT
Expression
,
Expression
,
Expression
PRINT
@
Expression
;
Expression
String Expression
,
;
PUT
Expression
String Expression
,
REPEAT
RENUMBER
Number
,
Number
REDUCE
0
1
2

20


SET
-
(
Expression
,
Expression
)
SPEED
Expression
,
Expression
TEST
Expression
,
Variable
TRACE
Expression
,
String Expression
VDU
Expression
,
Expression
,
String Expression
WHILE
Expression
WRAP
+
-
WEND
XLIST
Number
XREF
Number

21

Appendix 6: Nested Loops

There are 4 types of loops available in Extension Basic:
1) FOR ..... NEXT
2) REPEAT ... UNTIL
3) WHILE .... WEND
4) GOSUB ... RETURN

Any loop can be nested within any other loop. Loops of the same type can be nested, as can loops of different types.

However, when loops become un-matched (e.g as a result of using a GOTO to jump out of a loop) rules are needed to predict what will happen.

The rules are used whenever a NEXT, UNTIL, WEND or RETURN is found. They are:-

NEXT: If the matching loop on the stack is a FOR, behave as in Nascom BASIC. Otherwise, report a ?NF Error (Next without For).

UNTIL: Remove any FOR entries from the stack, closing any mismatched FOR...NEXT loops. Then, if the matching entry on the stack is not a REPEAT, report a ?UR Error (Until without Repeat). Otherwise, evaluate the expression after the Until, and if it is false loop back to the Repeat. Otherwise, remove the entry from the stack, closing the loop.

WEND: Remove any FOR entries from the stack. If the matching entry on the stack is not a WHILE, report a ?EW Error (End without While). Otherwise, branch back to the matching While.

RETURN: Remove all entries from the stack until a matching GOSUB is found. If it is not found, report a ?RG Error (Return without GOSUB). Otherwise, remove the entry from the stack and return to the statement after the Gosub.

Execution of WHILE ... WEND

The WHILE ... WEND loop is an unusual facility for an interpreter since it can be executed zero times if the condition is false. In this case the program branches to the matching WEND statement and must scan ahead to find it.

Example:
```
10 WHILE A<0
20 PRINT "A<0": A=A+1
30 WHILE B<0
40 PRINT "B<0": B=B+1
50 WEND
60 WEND
```

If the condition A<0 in line 10 were false, then the program continues after line 60.

In order that the interpreter can do this efficiently, WHILE and WEND keywords must be at the start of lines. Note that a failure to do this will not give rise to any error message, however.

22

Appendix 7: Workspace Addresses

If you wish to use machine code routines with Extension Basic, or run it using your own external I/O drivers you will need to know which areas of memory are used by Extension Basic as workspace.

The USR function and addresses 1003-1006 are not used by EB; 'USR's will work as in ROM BASIC.

Extension Basic sets up the following reflections (jumps) when initialised:

Name	Address	Purpose
$UIN	ØC7A	Used to 'trap' direct commands.
$IN	ØC75	Used by repeat keyboard.
$UOUT	ØC77	Used by WRAP statement.
SET	1Ø46	Used by SET statement.
EBSTRT	1ØØ1	Basic workspace. Used by EB warm start.

In addition locations ØF8Ø to ØFFF are used as workspace.

The stack occupies the top of memory, just below the area reserved for string space, in the same way as ordinary Nascom Rom Basic.

PRINTER ROUTINE : D000 TO DRIVE WALTERS
SEE INSIDE FRONT COVER

1. L ØDØØ
2. N TO BEAP
3. NO TO NASPEN
4. Y TO CLSBASIC
5. U - WARMSTART TO PRINT
6. WARM START BACK TO BASIC
7. "PRINT"

23

Appendix 8: Keyword Codes

This appendix should be used in addition to the Nascom Rom Basic manual appendix J; Single Character Input of Reserved Words.

This appendix lists the new keywords used by Extension Basic and gives their hex code and the single-character entry keys.

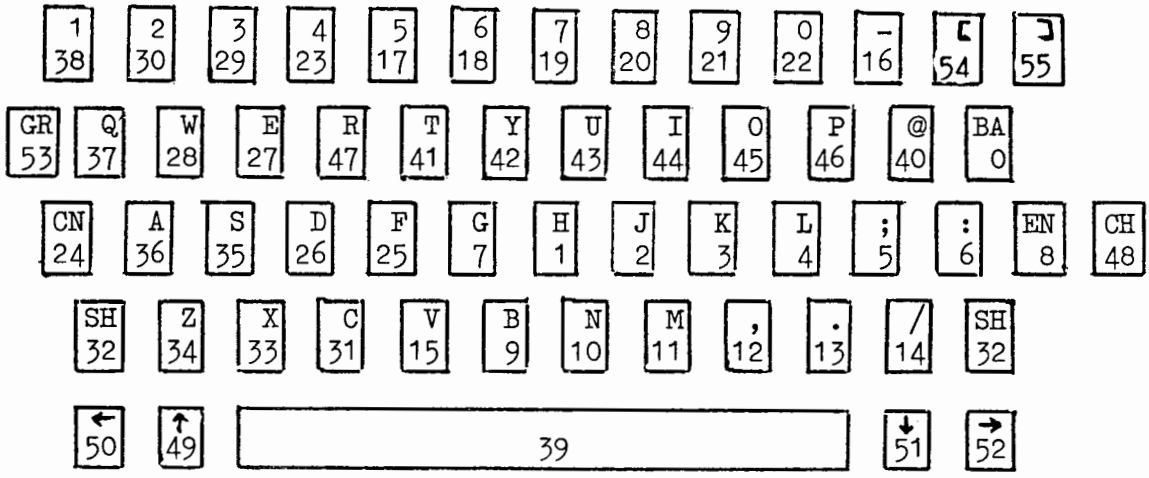
Reserved Word	Hex code	Key- Graphics plus:
DELAY	D0	P
PUT	D1	Q
CALL	D2	R
VDU	D3	S
DEC	D4	T
AUTO	D5	U
SPEED	D6	V
TRACE	D7	W
XREF	D9	Y
XLIST	DA	Z
INKEY	DB	⊏
GET	DC	\ (Shift ⊏)
TEST	DD	⊐
COPY	DE	↑ (Shift 0)
INLIN	DF	_ (underline)
BREAK	E0	Control space
ELSE	E1	a
LINE	E2	b
PLOT	E3	c
WRAP	E4	d
EDIT	E5	e
FIND	E6	f
DELETE	E7	g
REDUCE	E8	h
REPEAT	E9	i
UNTIL	EA	j
WHILE	EB	k
WEND	EC	l
RENUMBER	ED	m
CHECK	EE	n
HEX	EF	o

24

Appendix 9: Character Codes for TEST

The character codes used by the TEST statement, to specify which key is checked to see if it is up are down, are not ascii codes. Instead, they relate to the keyboard hardware and the way in which the keys are wired up.

The keyboard plan below relates these codes to the keys they represent:



Appendix 10: CALL argument details

The CALL statement is an alternatine to the USR statement, and simplifies parameter passing. Its format is:

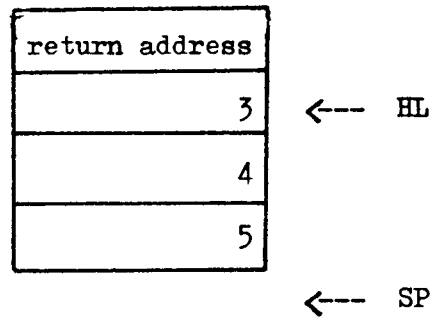
CALL address [,argument-list]

The address and optional arguments are expressions that evaluate to 2 byte integers (numbers in the range -32768 to +32767).

When a CALL statement is executed, the return address is pushed onto the stack followed by 2 byte values for the arguments. Register HL is set to the value of the stack pointer when the return address had just been pushed. Register C holds the number of arguments (zero and upwards).

The following example illustrates how values are positioned on the stack:

CALL 100,3,4,5



The subroutine at address should pop the argument values to use them, or if it is necessary to return without popping them, it should use the following sequence of instructions:

```
LD    SP,HL
RET
```

A routine without arguments can simply RET.

CALL does not require you to save and restore registers. You can change any registers without risk to the calling BASIC program.

26

Appendix 11: Adding Your Own Keywords

Extension Basic allows you to add extra keywords for statements or commands and thus extend BASIC even further.

When EB **inputs a** statement, it scans the ROM BASIC keyword table followed by the EB keyword table and finally **any** user-defined keyword table to compress each keyword to a single byte. When it subsequently interprets the statement it processes each keyword by branching to the address in the jump table corresponding to the keyword table,and position within it,for the byte value.

Thus, adding your own keywords involves defining a keyword table and providing code to handle the action of each keyword in it.

The following locations are relevant to defining a keyword table:

Address	Name	Description
F80-F81	BASNAM	; Address of ROM BASIC keyword table
F82-F83	EBNAM	; Address of Extension Basic keyword table
F84-F85	EXTNAM	; Address of External (user defined) keyword table
F86-F87	BASJP	; Address of ROM BASIC jump table (to keyword routines)
F88-F89	EBJP	; Address of Extension Basic jump table
F8A-F8B	EXTJP	; Address of External (user defined) jump table
F8C	NOEB	; Number of EB keywords
F8D	NOEXT	; Number of External (user defined) keywords

The structure of a keyword table is as follows:

```
Unused byte
First character (top bit set), remainder of keyword    : First keyword
:
:
:
First character (top bit set), remainder of keyword    : Last keyword
Byte with only the top bit set (hex 8Ø).
```

In parallel with the keyword table, a jump table is needed. This contains a 2-byte entry corresponding to each keyword in the table, containing the address of the code to handle each keyword. The structure is:

```
2-byte address of code to handle first keyword
:
:
:
2-byte address of code to handle last keyword.
```

To define and use a keyword table and the corresponding jump table, you should set one up and change EXTNAM to hold the address of the keyword table, EXTJP to the address of the jump table and NOEXT to hold the number of keywords. Note that, due mainly to the profligacy of ROM BASIC, you can only define 16 keywords and these will have values F0 to FF in the BASIC program. You could effectively define more keywords by using the first parameter to define the function, with each keyword representing a range of functions.

Note: EB cold start initialises all the locations named above, so you will need a 'cold start' utility which sets EXTNAM, EXTJP, and NOEXT and which you execute after cold starting EB each day.

When a keyword routine is called, the following registers are set up:

HL address of the first non-space character after the keyword
A character (HL)
Flags:
Z set if A holds Ø or ':', i.e if the keyword is at the end of the line
C set if A holds a numeric digit

The keyword routine should carry out any processing and then return to EB using a RET. When it does so, HL must hold the address of a line terminator (Ø or ':') or a ?SN error will result. Thus keyword code which starts with RET NZ will validate that no arguments were given; producing a ?SN error if they were.

All registers apart from HL can be corrupted with impunity.

The following routines within ROM BASIC will be of use in writing keyword routines:

Address	Name	Description
E3AD	SNERR	Prints ?SN Error in line nn and stops.
E3BF	TMERR	Prints ?TM Error in line nn and stops.
E3F8	PRTOK	Prints Ok and stops.
E68A	CPHLDE	Compares HL and DE. Z is set if HL=DE. C is set if HL<DE. HL and DE are unchanged.
E690	CALL CPHL DEFB COMMA	If (HL) is the same as the byte following the CALL, call SEARCH, add 1 to return address and return. Otherwise report a syntax error. Use to skip commas/brackets.
E836	SEARCH	Find first non-space character (HL). Increment HL, check if (HL) is a space and repeat until it is not. A=(HL), set C if A is numeric, set Z if A is a line terminator.
E9AØ	FCERR	Prints ?FC Error in line nn and stops.
E9A5	GETNO2	DE = numeric constant (HL): 0-65529. (HL) is next non-space.
EA46	ULERR	Prints ?UL Error in line nn and stops.
F484	GETNO1	DE = numeric expression (HL): 0-255. (HL) is next non-space.

The following routine in Extension Basic will also prove useful:

Address	Name	Description
EB + 20C	GETNO3	DE = numeric expression (HL): -32768 - 32767. (HL) is next non-space character. The EB in the address = EB start address.

Commercial: If anyone produces a package of useful routines to enhance EB, via this mechanism, Level 9 Computing is prepared to market it: circulating details to buyers of Extension Basic. There is clearly a need for such a package; perhaps providing routines for bulk data storage on cassette etc.

28

Appendix 12: Hints on using Extension Basic

A number of features of EB may not be obvious from the manual and some of the more interesting are described below:

1) TRACE takes a string expression as its second argument. If, for example, you just wanted to trace one section of your program, you could insert TRACE 1,T$ at the start of the section, TRACE Ø at the end. Then by entering T$=trace-string before RUNning the program, just the required section would be traced. This also allows you to slow down execution of a particular piece of code...

2) TRACE can include constant text in its output, by means of using the CHR$ function in its string expression to enter quotes. For example:

```
TRACE 1,CHR$(34)+"A="+CHR$(34)+"A"
```

will produce output of the form:

```
1Ø       A= 24
```

3) Surrounding a section of code by BREAK - and BREAK + prevents the use of ESC to stop the program. This is useful to protect critical sequences (e.g cursor addressing or updating a complex data structure) being half completed when the program is stopped.

4) Delays due to TRACE and DELAY cannot be interrupted by ESC, and DELAY Ø causes a very long delay indeed (due to the way the end condition is coded).

5) RAM Extension Basic checksums its code on cold-start, to guard against possible overwriting by user-programs. In ROM EB, this code is replaced by a routine of the same size exactly, in the same place, which enables the power-on-reset jump to be used to automatically initialise EB. A checksum error is indicated by a message: Error.

6) SET - as the first statement of a program automatically restarts EB.

7) WRAP only functions properly with WIDTH set to 255, otherwise you get messy extra new-lines.

8) FIND works by doing a case-insensitive match between the packed input string and the packed program. Thus it is very fast and matches upper/lower case. However, in some unusual cases, FIND will also pick up extra strings in addition to those that it should. This may never happen to you, but if it does, don't worry.

9) Append. Extension Basic does not have an append command for adding one program onto the end of another. However, the effect of APPEND can be achieved as follows, provided the lines in the program to be appended are no wider than the screen.

- RENUMBER the program to be appended with large line numbers:
- Save it to cassette using the commands:

```
NULL 255          - add a delay after each line;
WIDTH 255         - prevent spurious CRs;
LINES 32ØØØ       - turn off paging;
MONITOR
XØ                - direct listing to cassette;
Z
SET               - restart EB;
LIST              - list to tape (start it on RECORD first!);
```

- Use RESET, Z and SET to clear "XØ" mode and return to EB;
- Load the program that you want to append onto;
- Rewind the tape created above and simply play it. EB adds characters from this as if they came from the keyboard - carrying out the APPEND.